Fractional Physics-informed Neural Networks

Pavan Pranjivan Mehta[1]

[1]SISSA, International school of Advanced Studies. Email : pavan.mehta@sissa.it, Website : www.pavanpmehta.com

Presented on 19 June 2024 at **Scientific Machine Learning, emerging topics, Trieste, Italy** (18-21 June 2024)

Abstract

Fractional calculus is a generalization of classical (integer-order) derivatives, where the order can be arbitrary. Paradoxically, these are non-local operators, addressing anomalous diffusion processes. Recently, fractional derivatives have found applications in real world phenomena, such as turbulence (Mehta et. el, 2019; Mehta, 2023). In order to solve for fractional differential equations, fractional physics-informed neural networks (fPINNs) were introduced in Pang et el. (2019) for forward problems. This was further extended for inverse problems in Mehta et. el. (2019) and furthered in Mehta (2023) for learning the by developing pointwise fPINNs and extended to tempered definitions.
It is to be noted that the underlying stochastic process governed by fractional operators has an infinite second moment, which is not the case for real world scenarios. Thus, a tempered definition was introduced producing finite second moments; also truncated definitions, which not only has finite second moment but also addresses the computational challenges. Thus, in this presentation, I will also present the algorithms developed in Mehta (2023) for application to both these definitions and numerically determine the equivalence between these two operators. As a result, I will introduce and compute the horizon of non-local interactions.

References

Mehta, P. P. (2023). Fractional and tempered fractional models for Reynolds-averaged Navier–Stokes equations. Journal of Turbulence, 24(11-12), 507-553.
Mehta, P. P., Pang, G., Song, F., & Karniadakis, G. E. (2019). Discovering auniversal variable-order fractional model for turbulent Couette flow using a physics-informed neural network. Fractional calculus and applied analysis, 22(6), 1675-1688.
Pang, G., Lu, L., & Karniadakis, G. E. (2019). fPINNs: Fractional physics-informed neural networks. SIAM Journal on Scientific Computing, 41(4), A2603-A2626

Acknowledgment

This research was initiated and carried out at Brown University, USA with the mentorship of Prof. George Karniadakis. Research was supported using computational resources and services at the Center for Computation and Visualization, Brown University.
Funding for this conference is supported by PNRR digitalization, Italy by generosity of Prof. Gianluigi Rozza (SISSA).

Introduction to Fractional Calculus

4.1 Riemann-Liouville Definition

Recall the Cauchy's formula for repeated integration (1) for $p \in \mathbb{N}$,

$$\underbrace{\int_a^x \int_a^{x_p} \int_a^{x_{p-1}} \cdots \int_a^{x_2}}_{\text{p-integrals}} f(x_1) dx_1 dx_2 \dots dx_p = \frac{1}{(p-1)!} \int_a^x (x-\tau)^{p-1} f(\tau) d\tau \quad (1)$$

Here, $p \in \mathbb{N}$, to generalise this formula (1) introduce $\Gamma(p) = (p-1)!$, where $\Gamma(.)$ is the Euler gamma function, thus the Riemann-Liouville fractional integral is defined as (2) for ($p \in \mathbb{R}_+$)

$$_aI_x^p f(x) \ := \ \frac{1}{\Gamma(p)} \int_a^x (x-\tau)^{p-1} f(\tau) d\tau \quad (2)$$

Subsequently, the Riemann-Liouville (RL) fractional derivative is defined as (3) for $p \in \mathbb{R}_+$ and $k \in \mathbb{N}$,

$$^{RL}_{\ a}D_x^p f(x) \ := \ \frac{1}{\Gamma(k-p)} \frac{d^k}{dx^k} \int_a^x (x-\tau)^{k-p-1} f(\tau) d\tau \ \ , \ k-1 \le p < k \quad (3)$$

here, (d^k/dx^k) is the classical integer-order derivative.

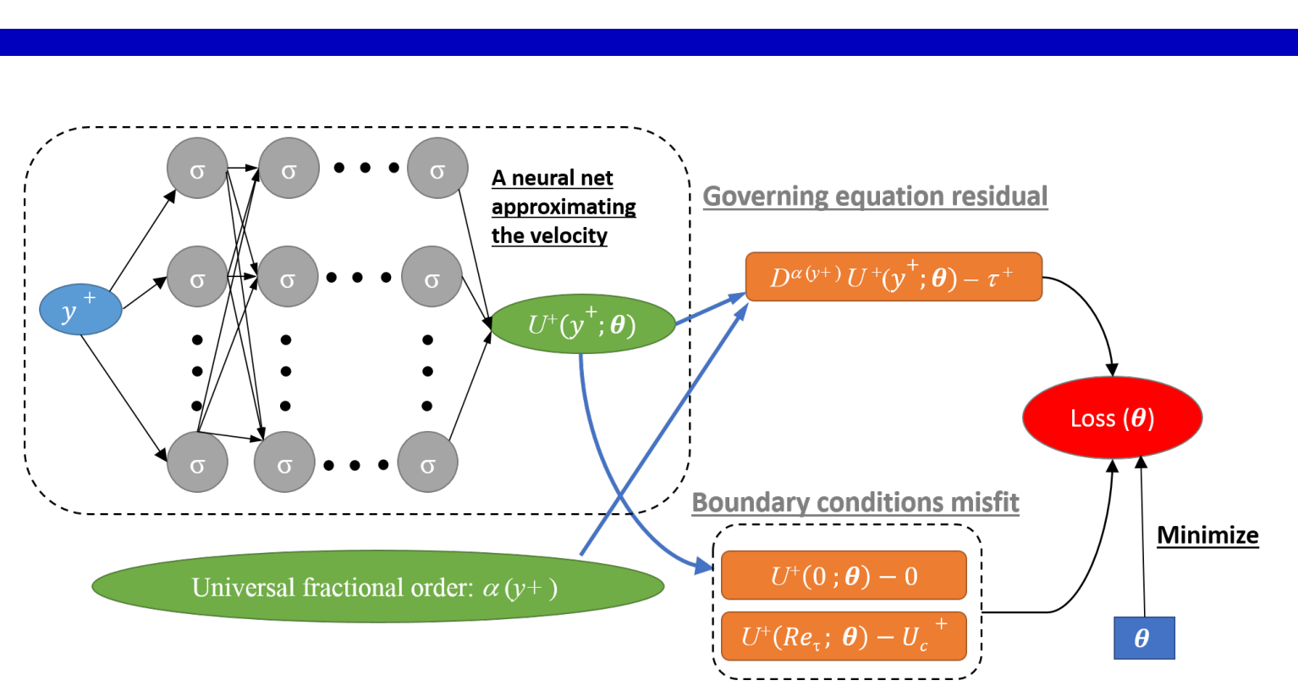
4.2 Caputo Definition

Although, the RL derivative is mathematically well established, there are two key problems **(a)** for a constant function RL derivative is non-zero **(b)** It's problem with specifying initial conditions.
As a remedy, Caputo defined a fractional derivative as (4) for $p \in \mathbb{R}_+$ and $k \in \mathbb{N}$,

$$^{C}_{a}D_x^p f(x) \ := \ \frac{1}{\Gamma(k-p)} \int_a^x (x-\tau)^{k-p-1} \frac{d^k}{dx^k} f(\tau) d\tau \ \ , \ k-1 < p \le k \quad (4)$$

Remark 1. For homogeneous conditions the Caputo fractional derivative is equivalent to the Riemann-Liouville fractional derivative.

Forward Problem (Mehta et el., 2019)


A neural net approximating the velocity
y^+
σ
$U^+(y^+;\theta)$
Governing equation residual
$D^{\alpha(y^+)}U^+(y^+;\theta) - \tau^+$
Loss (θ)
Boundary conditions misfit
$U^+(0;\theta) - 0$
$U^+(Re_\tau;\theta) - U_c^+$
Minimize
θ
Universal fractional order: $\alpha(y^+)$

Given the fractional order $\alpha(y^+)$ and the boundary conditions, we aim to solve $D_{y^+}^{\alpha(y^+)}U^+ = 1$ for the mean-flow velocity $U^+(y^+)$ for $y^+ \in (0, \mathrm{Re}_\tau]$. We approximate $U^+(y^+)$ by a multi-layer feedforward neural network $U_{NN}^+(y^+;\boldsymbol{\theta})$. To determine the parameters $\boldsymbol{\theta}$, we minimize the following loss function with respect to $\boldsymbol{\theta}$

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[D^{\alpha(y_i^+)}U_{NN}^+(y_i^+;\boldsymbol{\theta}) - 1\right]^2 + (U_{NN}^+(0;\boldsymbol{\theta}) - 0)^2 + (U_{NN}^+(\mathrm{Re}_\tau;\boldsymbol{\theta}) - U_c^+)^2, \ y_i^+ \in (0, \mathrm{Re}_\tau]. \quad (5)$$

Given a fixed integer M, we employ the L_1 scheme to approximate the fractional derivative in the loss function

$$D^{\alpha(y_i^+)}U_{NN}^+(y_i^+;\boldsymbol{\theta}) \approx \frac{h(y_i^+)^{-\alpha(y_i^+)}}{\Gamma(2-\alpha(y_i^+))}\sum_{k=0}^{M-1} c_{M,k}\left(U_{NN}^{+,k+1} - U_{NN}^{+,k}\right), \quad (6)$$

where $c_{M,k} = (M-k)^{1-\alpha(y_i^+)} - (M-k-1)^{1-\alpha(y_i^+)}$, $U_{NN}^{+,k} := U_{NN}^+(kh(y_i^+))$, and $h(y_i^+) = y_i^+/M$. Upon obtaining the optimal $\boldsymbol{\theta}$ we can predict the velocity profile at any location $y^+ \in (0, \mathrm{Re}_\tau]$ using $U_{NN}^+(y^+;\boldsymbol{\theta})$.

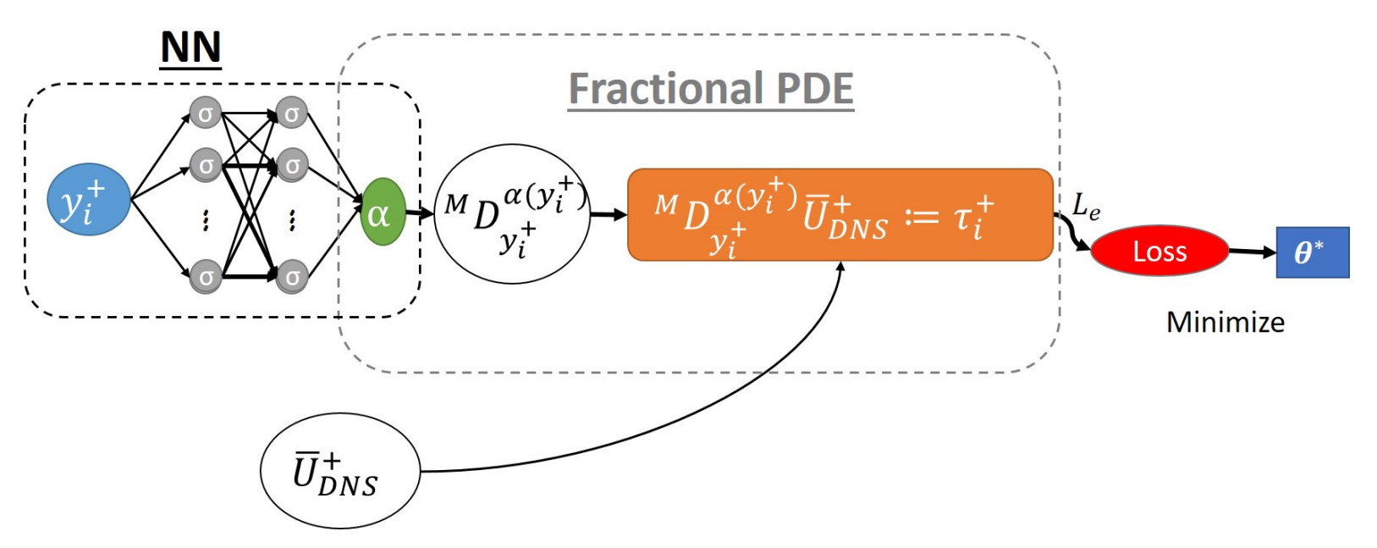
Inverse Problem (pointwise) (Mehta, 2023)

6.1 Inverse problem: computing fractional order (pointwise) (Mehta, 2023)

For the inverse problem the goal is to find the fractional order ($\alpha(y^+)$) given the function $\overline{U^+}$ and τ^+ satisfying (7)

$$0.5\left({}_0^C D^{\alpha(y^+)} - {}_{y^+}^{\ C} D^{\alpha(2Re_\tau)}\right)\overline{U^+} \ = \ \tau^+, \ y^+ \in (0, Re_\tau) \quad (7)$$

Here we developed a **pointwise strategy** (Mehta, 2023), this implies that we train the neural network for a single training point at a time.


NN
y_i^+
σ
α
Fractional PDE
$^M D_{y_i^+}^{\alpha(y_i^+)}$
$^M D_{y_i^+}^{\alpha(y_i^+)} \overline{U}_{DNS}^+ := \tau_i^+$
L_e
Loss
θ^*
Minimize
$\overline{U}_{DNS}^+$

The input of feed forward neural network is single location y_i^+ (only one training point), where $i = \{1, 2, \dots, N\}$ for N training points. Indeed, the fractional order (the output of neural network) is evaluated that location (y_i^+). The loss function only comprises of the equation term (L_e) given in (8) written for the i^{th} training point.

$$Loss = L_e = \left[{}^M D_{y_i^+}^{\alpha_{NN}(y_i^+)}(\overline{U_{DNS}^+}) - \tau^+(y_i^+)\right]^2 \quad (8)$$

For numerical discretisation of the fractional derivative we employ an *L1* scheme (Mehta, 2023) for the left- and right-sided derivative.
The merit of pointwise strategy are summarised as follows (Mehta, 2023):

- Given that the fractional order exists, it does not require boundary conditions, which earlier required to impose unity in Mehta et el. (2019). Thus for more complex situations we can find a physically consistent fractional order.
- There are cases, where the fractional order is a discontinuous function or the neural network collapses during training process; a pointwise algorithm mitigates all the limitations. However, for forward problem, we rarely encounter any problems during its training as demonstrated in Mehta et el. (2019).

Algorithms for tempered and truncated fractional definitions (Mehta, 2023)

7.1 Investigating the influence of tempering parameter over fractional order (Mehta, 2023)

In the first investigation, we evaluate the influence of fractional order for tempered or truncated fractional two-sided model, associated with the tempering parameter λ and δ^+, respectively. Algorithm 1 is the modified pointwise fractional physics-informed neural network algorithm, where we additionally supply the tempering parameter λ or δ^+. Please note, again for this pointwise algorithm, NO boundary conditions are required for solving the inverse problem of determining the fractional order.
The left- and right- sided tempered fractional derivative as (9) and (10), respectively.

$$\begin{aligned} {}_0^C D_{y_i^+}^{(\alpha(y_i^+),\lambda)}(\overline{U^+}) &= \frac{1}{\Gamma(1-\alpha_i)}\int_0^{y_i^+}(y_i^+ - \zeta)^{-\alpha_i} e^{-\frac{\lambda}{Re_\tau}(y_i^+ - \zeta)}(\overline{U^+}(\zeta))' d\zeta \\ &= -\frac{h^{-\alpha_i}}{\Gamma(2-\alpha_i)}\sum_{j=0}^{l-1}(\overline{U^+}_{l-j} - \overline{U^+}_{l-j-1})e^{-\frac{\lambda}{Re_\tau}(jh_l)}\left[(j+1)^{1-\alpha_i} - j^{1-\alpha_i}\right] \end{aligned} \quad (9)$$

$$\begin{aligned} {}_{y_i^+}^{\ C} D_{2Re_\tau}^{(\alpha(y_i^+),\lambda)}(\overline{U^+}) &= \frac{-1}{\Gamma(1-\alpha_i)}\int_{y_i^+}^{2Re_\tau}(\zeta - y_i^+)^{-\alpha_i} e^{-\frac{\lambda}{Re_\tau}(\zeta - y_i^+)}(\overline{U^+}(\zeta))' d\zeta \\ &\approx \frac{-h_l^{-\alpha_i}}{\Gamma(2-\alpha_i)}\sum_{j=0}^{M-l-1}(\overline{U^+}_{l+j} - \overline{U^+}_{l+j+1})e^{-\frac{\lambda}{Re_\tau}(jh_l)}\left[(j+1)^{1-\alpha_i} - (j)^{1-\alpha_i}\right] \end{aligned} \quad (10)$$

The loss function for the feed forward neural network uses the discrete form for the left- (9) and right-sided (10) definition.

$$Loss = L_e = \left[\frac{1}{2}\left({}_0^C D_{y_i^+}^{(\alpha(y_i^+),\lambda)}(\overline{U^+}) \ - \ {}_{y_i^+}^{\ C} D_{2Re_\tau}^{(\alpha(y_i^+),\lambda)}(\overline{U^+})\right) \ - \ \tau^+(y_i^+)\right]^2 \quad (11)$$

For the truncated definition, we employ the same finite difference scheme, over the same grid for truncated left-sided as (12) and right sided as (13). The only difference the domain, for the fractional derivative the domain was $[0, 2Re_\tau]$, while for the truncated definition the domain is $[y^+ - \delta^+, y^+ + \delta^+]$, where δ^+ is a composite function (refer Mehta, 2023) and $y^+ \in (0, 2Re_\tau)$.

$$\begin{aligned} &{}_0^C D_{y^+}^{(\alpha(y_i^+),\delta^+)}(\overline{U^+}) \approx \\ &-\frac{h_k^{-\alpha(y_i^+)}}{\Gamma(2-\alpha(y_i^+))}\sum_{j=0}^{k-1}(\overline{U^+}_{k-j} - \overline{U^+}_{k-j-1})[(j+1)^{1-\alpha(y_i^+)} - j^{1-\alpha(y_i^+)}] \end{aligned} \quad (12)$$

$$\begin{aligned} &{}_{y^+}^{\ C} D_{2Re_\tau}^{(\alpha(y_i^+),\delta^+)}(\overline{U^+}) \approx \\ &-\frac{h_k^{-\alpha(y_i^+)}}{\Gamma(2-\alpha(y_i^+))}\sum_{j=0}^{N-k-1}(\overline{U^+}_{k+j} - \overline{U^+}_{k+j+1})[(j+1)^{1-\alpha(y_i^+)} - j^{1-\alpha(y_i^+)}] \end{aligned} \quad (13)$$

The loss function for the feed forward neural network uses the discrete form for the left- (12) and right-sided (13) definition, given a truncating parameter δ^+.

$$Loss = L_e = \left[\frac{1}{2}\left({}_0^C D_{y^+}^{(\alpha(y_i^+),\delta^+)}(\overline{U^+}) \ - \ {}_{y^+}^{\ C} D_{2Re_\tau}^{(\alpha(y_i^+),\delta^+)}(\overline{U^+})\right) \ - \ \tau^+(y_i^+)\right]^2 \quad (14)$$

Algorithm 1: Pointwise computation of fractional order $\alpha(y^+)$ for tempered or truncated fractional two-sided model

```
Data: Domain, $\Omega = [0, 2Re_\tau]$
Data: Velocity, $\overline{U}^+(y^+)$
Data: Fix tempering or truncating parameter, $\xi = \{\lambda, \delta^+\}$
Result: $\alpha(y^+)$
Function compute_residual($y^+$, $\overline{U}^+(y^+)$, $\xi$):
    // Compute left-sided derivative for a fixed $\xi$
    $D_l^\xi \leftarrow$ Use (9) for tempered and (12) for truncated formulation ;
    // Compute right-sided derivative for a fixed $\xi$
    $D_r^\xi \leftarrow$ Use (10) for tempered and (13) for truncated formulation ;
    // Residual for two-sided model
    $residual \leftarrow 0.5\ (D_l^\xi - D_r^\xi)$ ;
return $residual$
Function compute_fractional_order($y^+$, $\overline{U}^+(y^+)$, $\xi$):
    // Computes $\alpha$ for supplied $y^+$
    // Given: a fixed $\xi$ and $\overline{U}^+(y^+)$
    initialize neural network ;
    while $loss > threshold$ do
        // get residual value of equation
        $residual \leftarrow$ compute_residual($y^+$, $\overline{U}^+(y^+)$, $\xi$) ;
        $loss \leftarrow residual$;
        train neural network;
    end
return $\alpha$
Main:
    $y\_train \leftarrow$ discrete points in $(\Omega)$;
    for $y^+ \in y\_train$ do
        // iterates over the list $y\_train$
        $\alpha(y^+) \leftarrow$ compute_fractional_order($y^+$, $\overline{U}^+(y^+)$, $\xi$) ;
    end
End
```

7.2 Horizon of non-local interactions and finding the equivalence between the truncated and tempered fractional derivative (Mehta, 2023)

Recognise, the parameter δ^+ associated with the truncated definition is also the horizon of non-local interactions.

Definition 7.1. The horizon of non-local interactions (δ^+) is a *distance beyond which no long-range interactions occur.*

If there exist such a distance δ^+, such that the fractional derivative defined over a infinite (or semi-infinite) domain is equivalent to its truncated definition with a compact support, then (15) holds. Following our notations, ${}_{x-\delta^+}{}^C D_x^\alpha f(x) \ = \ {}_{-\infty}{}^C D_x^{(\alpha,\delta^+)} f(x)$.

$$\left|{}_{-\infty}{}^C D_x^\alpha f(x) \ - \ {}_{x-\delta^+}{}^C D_x^\alpha f(x)\right| < \epsilon \quad (15)$$

Equivalence of tempered and truncated definitions: In order to compute δ^+, we start with the tempered definition, compute the fractional order for a given λ, then use this fractional order to compute the parameter δ^+ (of truncated definition) by gradually increment δ^+ until (16) or (17) holds. It is NOT assumed a-prior that δ_l^+ and δ_r^+ are a constant function, but treated as spatially varying function, as we find the δ_l^+ and δ_r^+ in a pointwise sense. For δ^+ being constant function or $\delta_l^+ = \delta_r^+$ are just a special case of our consideration.

$$\left|\ {}_0{}^C D_{y^+}^{(\alpha,\lambda)} f(x) \ - \ {}_0{}^C D_{y^+}^{(\alpha,\delta^+)} f(x)\ \right| < \epsilon \quad (16)$$

$$\left|\ {}_{y^+}{}^C D_{2Re_\tau}^{(\alpha,\lambda)} f(x) \ - \ {}_{y^+}{}^C D_{2Re_\tau}^{(\alpha,\delta^+)} f(x)\ \right| < \epsilon \quad (17)$$

Algorithm 2: To find δ^+ of truncated f-RANS by supplying α of tempered f-RANS for a given λ.

```
Data: Domain, $\Omega = [0, 2Re_\tau]$
Data: Velocity, $\overline{U}^+(y^+)$
Data: Get $\alpha(y^+)$ for a given $\lambda$ of tempered f-RANS from Algorithm 1
// $\delta_l^+(y^+)$ and $\delta_r^+(y^+)$ are associated with left- (12) and
   right-sided (13) truncated fractional derivative,
   respectively.
Result: truncating parameter, $\delta_l^+(y^+)$ and $\delta_r^+(y^+)$
Function compute_temp_fractional_der($y^+$, $\overline{U}^+(y^+)$, $\alpha$, $\lambda$):
    // Compute left-sided tempered derivative with fixed $\lambda$
    $D_l^\lambda \leftarrow$ Use (9) for left-sided tempered derivative derivative ;
    // Compute right-sided tempered derivative with fixed $\lambda$
    $D_r^\lambda \leftarrow$ Use (10) for right-sided tempered derivative derivative ;
return $D_l^\lambda$, $D_r^\lambda$
Function compute_truncated_left_der($y^+$, $\overline{U}^+(y^+)$, $\alpha(y_i^+)$, $\delta_l^+$):
    // Compute left-sided derivative for the supplied $\delta_l^+$
    $D_l^{\delta_l^+} \leftarrow$ Use (12) for left-sided truncated formulation ;
return $D_l^{\delta_l^+}$
Function compute_truncated_right_der($y^+$, $\overline{U}^+(y^+)$, $\alpha(y_i^+)$, $\delta_r^+$):
    // Compute right-sided derivative for the supplied $\delta_r^+$
    $D_r^{\delta_r^+} \leftarrow$ Use (13) for right-sided truncated formulation ;
return $D_r^{\delta_r^+}$
Main:
    set default $threshold$, $\delta_l^+$ and $\delta_r^+$ ;
    for $y_i^+ \in y\_train = discrete\text{-}points\ (\Omega)$ do
        // Get the value for left- and right-sided tempered
           fractional derivative
        $D_l^\lambda$, $D_r^\lambda \leftarrow$ compute_temp_fractional_der($y^+$, $\overline{U}^+(y^+)$, $\alpha(y_i^+)$,
         $\lambda$);
        while $error > threshold$ do
            // This while-loop computes the min. $\delta_l^+$
            // Get the value for left-sided truncated
               fractional derivative
            $D_l^{\delta_l^+} \leftarrow$ compute_truncated_left_der($y^+$, $\overline{U}^+(y^+)$, $\alpha(y_i^+)$,
             $\delta_l^+$) ;
            $error \leftarrow |D_l^\lambda - D_l^{\delta_l^+}|$ or $|D_l^\lambda - D_l^{\delta_l^+}|/|D_l^\lambda|$;
            increment $\delta_l^+$ ;
        end
        while $error > threshold$ do
            // Similarly for the right definition, compute min.
               $\delta_r^+$
            $D_r^{\delta_r^+} \leftarrow$ compute_truncated_right_der($y^+$, $\overline{U}^+(y^+)$, $\alpha(y_i^+)$,
             $\delta_r^+$) ;
            $error \leftarrow |D_r^\lambda - D_r^{\delta_r^+}|$ or $|D_r^\lambda - D_r^{\delta_r^+}|/|D_r^\lambda|$;
            increment $\delta_r^+$ ;
        end
    end
End
```